Teaching computing in statistical theory courses[*]

David R. Hunter
Department of Statistics
The Pennsylvania State University

June 27, 2005

Abstract

Graduate-level instruction in statistical computing need not be relegated to courses devoted to statistical computing; even theoretical statistics courses can include considerable computational content through well-designed exercises. Such exercises should be chosen so that they force students to learn essential computing skills without impeding instruction of theory. A series of examples used in a course on large-sample theory illustrates this thesis.

Key Words: Large-sample theory, Statistical computing

1 Introduction

There is little question that computing is an essential aspect of statistics, and graduate education in statistics must adapt to this reality. (Although graduate education is the focus of this article, the approach here can also be applied, with creativity, to undergraduate education.) Probably the most obvious way to train graduate students in computing is by following the advice of Gentle (2004), who states that "…all graduate programs in statistics should offer at least one course in statistical computing…" However, such a course is not the topic of this article.

Instead, this article proposes and illustrates a more general philosophy of computational statistics education, namely, that computing topics should be sprinkled throughout the statistics

[*]This article arose from an invited session at JSM 2004 on teaching computational statistics, organized by Jim Albert and featuring talks by Jim Gentle, John Monahan, and the author. The author thanks Jim Albert, Murali Haran, and anonymous reviewers for helpful comments on early versions of the manuscript.

curriculum. In particular, computing should be used to verify or amplify theoretical results whenever doing so requires useful computing skills.

There are several reasons, both practical and pedagogical, for this philosophy. First, such an approach kills two birds with one stone: Students learn some computing in addition to theoretical subjects when they are forced to use the former in pursuit of the latter. This particular advantage can be blunted if too much attention is distracted by the computing at the expense of the theory; thus, the instructor should carefully choose examples that will maintain an appropriate balance and be cautious about allowing too many computing digressions. Second, there are pedagogical advantages of the approach that Gentle (2004) calls the "just-in-time" training model, in which certain skills — computing skills, in this case— are taught only when they are needed for some other purpose. I find that students are often more receptive to new material when it is presented as a means to some greater end rather than as an end in itself.

Third, students are exposed to a wider variety of perspectives on computing when they learn from several different instructors than when their computing instruction comes primarily from a single source. Fourth, this philosophy reflects the true role of computation in statistical research: Just as computing is (usually) a tool used by theoretical statisticians rather than an object of research itself, computing may be integrated into theoretical statistical courses rather than being taught solely in computing courses. Finally, this philosophy has an important practical advantage: It makes offering, even requiring, computing education within the statistics curriculum easier from a scheduling perspective than it would be if computing were taught solely in computing courses.

Before delving into the examples that comprise most of this article, a word about terminology is in order, particularly as it relates to the title of the article. Some authors draw a distinction between the terms "computational statistics" and "statistical computing". Generally, I understand the former to be a subset of the latter; the former includes theoretical topics in statistics that intrinsically involve computing (e.g., Markov chain Monte Carlo and EM algorithms), whereas the latter also includes statistical software and purely computational topics that happen to have relevance for statisticians. However, this distinction appears to be somewhat blurred in the literature, and in any event it is often unimportant. What *is* often important — in this article, for example — is the distinction between mere statistical software issues and the rest of statistical computing. This article is *not* about merely using statistical software to apply statistical theory to analysis of data. For while facility using statistical software to analyze data is a very useful skill for graduate students to possess, it is not sufficient to cover the range of computing skills that scientists often need to conduct basic research in statistics today.

The remainder of this article discusses embedding computing content into the large-sample theory course I taught at Penn State for several years. An old saying goes "A picture is worth a thousand words," and I think the pedagogical equivalent would be something like "an example is worth a thousand explanations." With this in mind, each of the next three sections presents and discusses a different example of a homework exercise based on assignments I have written in the asymptotics course. Some sample computer code is included as an appendix.

2 An inconsistent MLE

Exercise: *Define* $\delta \equiv \delta(\theta) = \exp\{-1/(1-\theta)^4\}$*. For* $\theta = .2$*, generate a simple random sample of size* $n = 50$ *from the density function*

$$f_\theta(x) = \theta g(x) + (1-\theta)\frac{1}{\delta}h\left(\frac{x-\theta}{\delta}\right), \tag{1}$$

where

$$\begin{aligned} g(x) &= 1/2 \quad \textit{for } -1 < x < 1 \\ h(x) &= 3(1-x^2)/4 \quad \textit{for } -1 < x < 1. \end{aligned}$$

Next, graph the log-likelihood function and find the MLE.

This exercise is based on an example due to Ferguson (1982) in which the maximum likelihood estimator is inconsistent. In fact, in this example, the MLE converges in probability to

1 as $n \to \infty$ no matter what the true value of θ is (students are asked to prove this fact as a separate exercise that involves no computing).

Computing knowledge is required even to begin the problem, since students must be able to generate from a mixture distribution and also to generate from the density $h(x)$. This latter task may be accomplished using a rejection method or by direct inversion of the cumulative distribution function $H(x) = (2 + 3x - x^3)/4$. Inverting the cdf to obtain

$$H^{-1}(x) = 2 \cos \left\{ \frac{4\pi}{3} + \frac{1}{3} \cos^{-1}(1 - 2x) \right\}$$

is an interesting activity in and of itself, though this is a bit too much of a digression for this course and I would typically offer the closed-form $H^{-1}(x)$ as a hint.

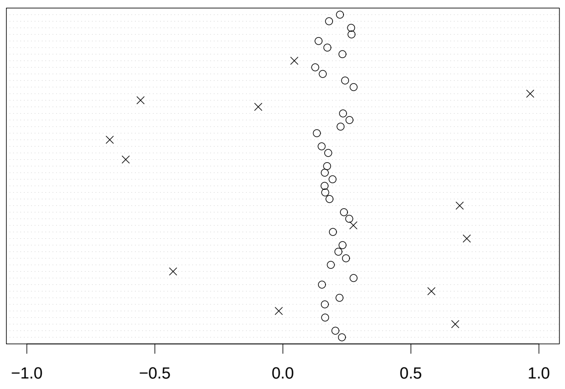

Figure 1: Random sample of size $n = 50$ from the density $f_\theta(x)$ of equation (1) with $\theta = 0.2$. The x's are uniform on $(-1, 1)$, whereas the o's are from $h[(x - \theta)/\delta]/\delta$.

Figure 1 depicts a random sample generated according to the specifications in the exercise. The points selected from the $h[(x - \theta)/\delta]/\delta$ density, marked by o's, must all fall in the range $\theta \pm \delta(\theta)$, which is $(.113, .287)$ in this case. The points selected from the uniform $(-1, 1)$ density, marked by x's, must include at least one point larger than about 0.7 in order for the exercise to work well. The probability that this fails, or $(1 - .15\theta)^n = .22$ when $n = 50$ and $\theta = 0.2$, may be a bit too large for one's taste; thus, an instructor could use a larger value of n and/or a larger value of θ so that each student is nearly assured of obtaining an interesting plot.

Once the sample is selected, plotting the loglikelihood function appears to be a trivial exercise: The loglikelihood is

$$\ell(\theta) = \sum_{i=1}^{50} \log \left\{ \frac{\theta}{2} + \frac{I_i(1-\theta)}{\delta} h[(x_i - \theta)/\delta] \right\}, \qquad (2)$$

where $I_i = I\{|x_i - \theta| < \delta\}$ denotes the indicator that θ lies within δ of x_i. Unfortunately, naively plotting equation (2) for $0 < \theta < 1$ results in Figure 2(a). What happened to the graph for $\theta > 0.77$? The answer lies in the fact that $\delta(\theta)$ is specifically constructed so that for θ near 1, $(1-\theta)/\delta$ is extremely large — so large, in fact, that it results in an overflow, treated as $+\infty$ by the software used to create figure 2(a). Thus, even when $I_i = 0$ and the product $I_i(1-\theta)/\delta$ should be zero, the computer doesn't know how to handle $0 \times \infty$. Diagnosing this problem correctly requires a student to think about how computer software represents numbers, and it reinforces the fact that a mathematical formula like equation (2) is not the same as a numerical algorithm: $\log[\exp(x)]$ is not always equal to x in computer arithmetic.

The error of Figure 2(a) is easily corrected by the simple device shown in Figure 2(b): Instead of calculating the loglikelihood directly as in equation (2), the value of I_i is tested for each i and the term involving $(1-\theta)/\delta$ is only calculated when $I_i = 1$; in case $I_i = 0$, the ith summand of equation (2) is replaced by $\theta/2$. Theoretically, the two methods are identical, but the second method never runs into the $0 \times \infty$ problem because $(1-\theta)/\delta$ is never computed when $I_i = 0$.

Having thus sidestepped one thorny computational issue, we are faced with a second. The computer is unable to graph a smooth curve as truly smooth; rather, it must "connect the dots" between points that lie on the curve. By choosing these points to be very close together, the computer fools our eyes into interpreting a graph as a smooth curve. Ordinarily, there is no harm in this artifice. However, in this example, the true loglikelihood has features that are so incredibly narrow that they are missed by connecting even the most fine-meshed grid of points.

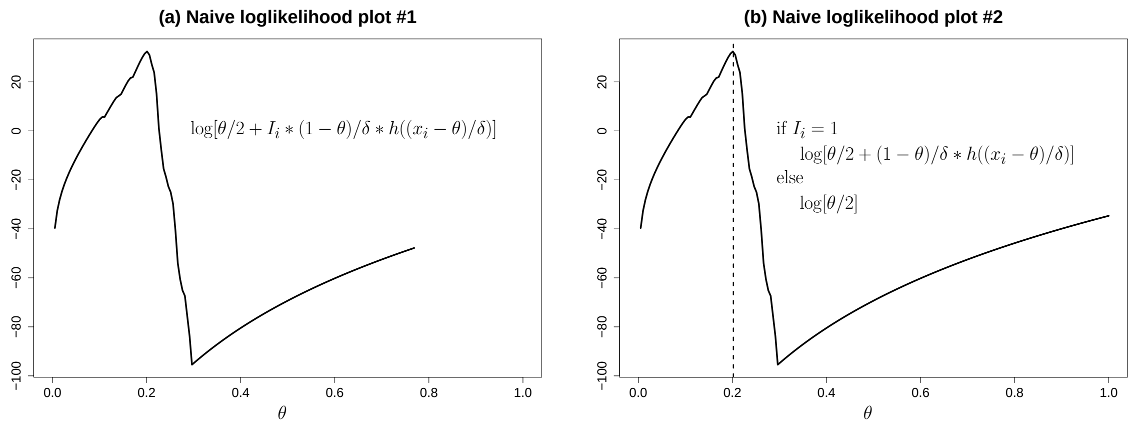

Figure 2: Plot (a) fails for $\theta > 0.77$, essentially due to the fact that the computer has not been told what to do with a number like $0 \times \infty$. The problem is corrected in plot (b), which uses a different algorithm that is mathematically but not computationally equivalent. Unfortunately, even plot (b) misses important features of the loglikelihood surface; in particular, the correct MLE is *not* 0.202 as it appears to be.

Bringing these features into view requires a slightly more detailed analysis.

Note that if $\theta = x_i$ for some sample point x_i, then $I_i = 1$ and $h[(x_i - \theta)/\delta] = 3/4$, so this point contributes

$$\log\left[\frac{\theta}{2} + \frac{3(1-\theta)}{4\delta}\right] \tag{3}$$

to the log-likelihood. As noted previously, $(1-\theta)/\delta$ can be enormous for θ close to 1, in which case expression (3) is approximately

$$\log\left[\frac{3(1-\theta)}{4\delta}\right] = \log 0.75 + \log(1-\theta) + \frac{1}{(1-\theta)^4}; \tag{4}$$

in fact, in 64-bit double-precision computer arithmetic, this "approximation" is exact for $\theta > .6$. Therefore, to correctly represent the loglikelihood and find the MLE, it is necessary to evaluate the loglikelihood at all $\theta = x_i$ specifically, and to use expression (4) when the naive approach in expression (3) results in overflow. In Figure 3, we see that the true MLE is equal to the largest sample value.

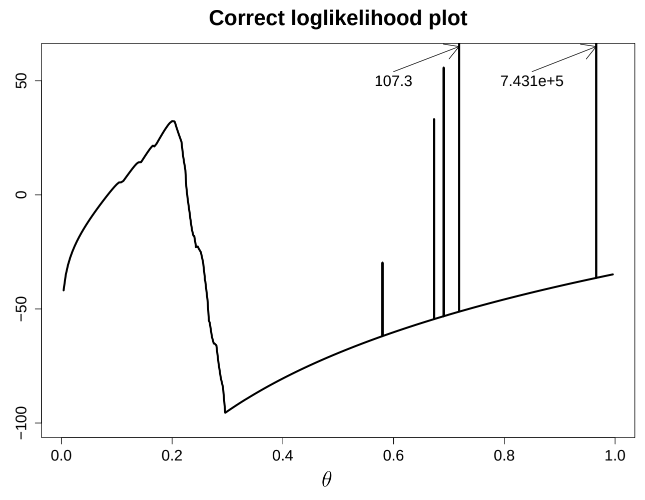

Figure 3: This plot shows that the MLE is the largest sample value, about 0.96.

There are further theoretical issues to which this exercise may lead. For instance, likelihood theory often guarantees the existence of a consistent root of the equation $\ell'(\theta) = 0$ even if the MLE is not a consistent root, and in this example the best likelihood-based estimator of θ turns out to be a local maximum other than the MLE. Also, from a Bayesian perspective, the correct likelihood in Figure 3 and the incorrect one in Figure 2(b) are essentially equivalent becuase the spikes in Figure 3 are so incredibly narrow that they have almost no impact on the posterior. Yet even as currently stated, the exercise illustrates an important theoretical result while simultaneously forcing students to confront several computing issues, including how to sample from a mixture distribution; how to sample from a distribution like $h(x)$; what can happen when we mean 0 but code it as $0 \times \infty$; the effect of overflow error and the fact that $\log[\exp(x)]$ is not always x in computer arithmetic; and how to produce a graph of a function.

3 Hardy-Weinberg equilibrium and Pearson's chi-square

Exercise: *There is a particular characteristic of human blood (the so-called MN blood group) that occurs in three distinct types: M, MN, and N. Under idealized circumstances called Hardy-*

Weinberg equilibrium, or HWE, these three types occur in a population with relative frequencies $p_1 = \pi_M^2$, $p_2 = 2\pi_M\pi_N$, and $p_3 = \pi_N^2$, respectively, where π_M is the relative frequency of the M allele in the population and $\pi_N = 1 - \pi_M$ is the frequency of the N allele. In a given sample, let n_1, n_2, and n_3 denote the number of individuals with the M, MN, and N types, respectively. If the value of π_M is not known, it may be estimated by the maximum likelihood estimator $\hat{\pi}_M = (2n_1 + n_2)/2n$, where $n = n_1 + n_2 + n_3$.

Consider the two null hypotheses

$$H_0^A : \pi_M = .3 \text{ and HWE holds;} \qquad H_0^B : \text{HWE holds.}$$

These hypotheses may be tested using the statistics

$$W^2 = \sum_{j=1}^{3} \frac{(n_j - np_j)^2}{np_j} \quad \text{and} \quad X^2 = \sum_{j=1}^{3} \frac{(n_j - n\hat{p}_j)^2}{n\hat{p}_j},$$

respectively, where $(\hat{p}_1, \hat{p}_2, \hat{p}_3) = (\hat{\pi}_M^2, 2\hat{\pi}_M\hat{\pi}_N, \hat{\pi}_N^2)$ and $(p_1, p_2, p_3) = (.09, .42, .49)$.

Simulate 1000 samples of size $n = 50$ from a population in which $\pi_M = .3$ and HWE holds. For each sample, compute both W^2 and X^2. Plot the empirical cumulative distribution functions for the 1000 values of W^2 and the 1000 values of X^2. Compare each graph with chi-squared distribution functions on 1 and 2 degrees of freedom. What do you conclude?

This is an example of a general type of computational exercise that is relatively easy to generate in an asymptotics course: A simulation study that checks an asymptotic result for a fixed sample size. This method of generating exercises is appealing since an elegant theoretical result can be all the more satisfying when it checks out empirically. However, because such exercises are easy to generate, there is a temptation to overproduce them, asking students to verify every new asymptotic result via simulation. This temptation should probably be avoided if only to preserve the students' sanity, though several carefully thought-out exercises of this type are definitely appropriate. It is particularly helpful if the different exercises are designed so that they cover a broad range of computing skills, rather than relying on the same limited set of skills again and again.

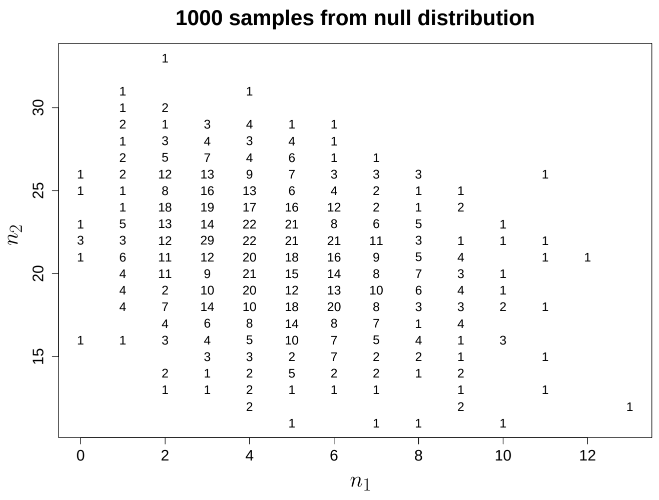

Figure 4: Shown are 1000 samples from multinomial$\{50, (.3^2, 2 \times .3 \times .7, .7^2)\}$. Only the values of n_1 and n_2 are shown; n_3 is always $50 - n_1 - n_2$. The plotted values are frequencies. For instance, there were 15 occurrences of $(5, 20, 25)$. Since $(5, 20, 25)$ results in $\hat{\pi}_M = .3$, the values of W^2 and X^2 coincide in this case and $W^2 = X^2 = 0.113$.

Figure 4 depicts 1000 simulated data points as called for in the exercise. Simulating the data amounts to simulating realizations from a multinomial distribution. Although many software packages will do this automatically, it is still worthwhile for students to know how to use the software to do it. And even if multinomial samples can be generated automatically, students might be encouraged to write their own multinomial-generating methods based on nothing but uniform pseudorandom numbers. Note that if true multinomial probabilities were used in place of simulated frequencies in Figure 4, the *exact* distributions of W^2 and X^2 could be computed; this too would be a useful computing exercise.

Once the multinomials are generated, the task remains to calculate 2000 chi-squared statistics. This clearly requires a bit of programming ability, though not a lot — after all, this is an asymptotics course, not a programming course. Doing this won't be difficult for students who already have some programming skill, and for those who don't it offers the chance to learn some simple programming techniques such as how to implement a loop, how to translate mathematical expressions into computer code, and how to store the results of multiple calculations for later use. I find that in the case of a student who has no idea how to begin such a task, offering a piece of example code performing a similar job is always sufficient to catalyze the student's successful completion of the task. In this example, explicit loops may be avoided entirely; see the sample code in the appendix.

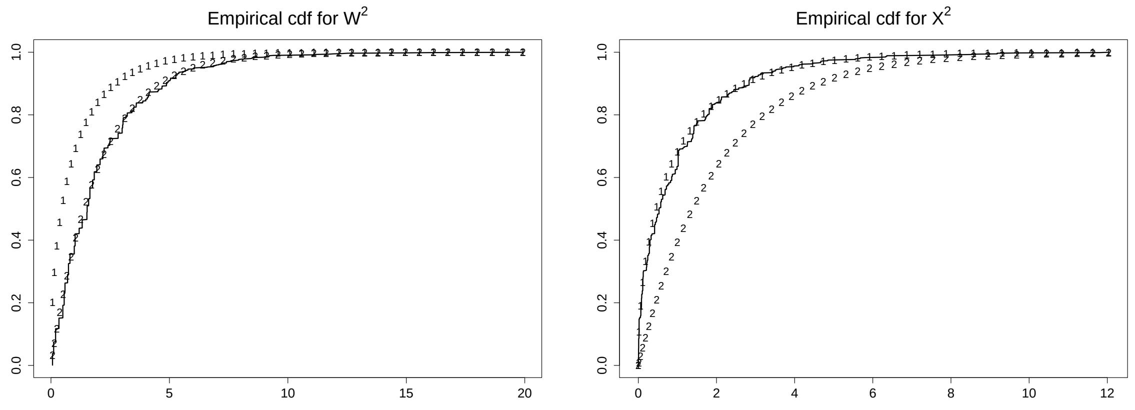

Figure 5: Shown in the left and right plots are the empirical cumulative distribution functions of 1000 realizations of W^2 and X^2, respectively. Superimposed on the graphs are the true cumulative distribution functions for χ^2_1 and χ^2_2 random variables, marked with 1's and 2's.

Once the W^2 and X^2 statistics are computed, students must grapple with the task of how to plot an empirical cdf, then how best to compare the result with chi-square distribution functions. Even obtaining the chi-square distribution functions is not a completely trivial exercise since this capability of statistical software is something with which students need to be familiar if they are not already. Figures 5(a) and 5(b) depict the empirical distribution functions of the W^2 and X^2 statistics, respectively. These figures powerfully illustrate the difference between the distribution of W^2 and the distribution of X^2 and how closely the asymptotic theory matches the finite-sample reality in this example.

It should be stressed that carrying out this exercise does not in any way serve as proof of the asymptotic distributions of W^2 and X^2, nor does it make such a proof less important in an asymptotics course. Computing should aid the instruction of theory, not supplant it.

4 Efficient one-step estimators

Exercise: *Generate 1000 random samples of size $n = 15$ from a logistic distribution with distribution function $F_\theta(x) = 1/(1 + \exp\{\theta - x\})$. Use $\theta = 2$. For each sample, calculate the median $\tilde{\theta}$, the maximum likelihood estimator $\hat{\theta}$, and the one-step Newton-method estimator based on $\tilde{\theta}$, namely*

$$\delta = \tilde{\theta} - \frac{\ell'(\tilde{\theta})}{\ell''(\tilde{\theta})}, \qquad (5)$$

where $\ell(\theta)$ is the loglikelihood function and $\ell'(\theta)$ and $\ell''(\theta)$ are its first and second derivatives. Find the sample variances of $\tilde{\theta}$, $\hat{\theta}$, and δ; how well do they match the asymptotic theory?

Like the previous exercise, this one is of the “check asymptotic results for finite samples” type. In this case, however, the theory itself suggests a computational topic: The asymptotic efficiency of so-called one-step estimators, like δ above, necessitates a discussion of Newton’s method at the very least. And since Newton’s method is such a commonly used computational tool, it is only natural to devise an exercise so that students are required to implement it.

A bit of asymptotics background might be helpful here. The maximum likelihood estimator in this and many other problems has the desirable characteristic that it is asymptotically efficient, which is to say that its asymptotic variance is in a certain sense as small as possible. (Note: this does not imply that the MLE has the minimum possible finite-sample variance.) The sample median, which is also a $\sqrt{n}$-consistent estimator in the sense that $\sqrt{n}(\tilde{\theta} - \theta)$ is bounded in probability, has a larger asymptotic variance than the MLE. To be precise, if n denotes the sample size, then an asymptotic approximation tells us that the MLE $\hat{\theta}$ is roughly normally distributed with variance $3/n$, whereas the median $\tilde{\theta}$ is roughly normally distributed with variance $4/n$. Incidentally, this problem could just as easily have been written using any other $\sqrt{n}$-consistent estimator of θ in place of (or in addition to) the sample median. For instance, we may verify using the central limit theorem that $\overline{X}$ is approximately normally distributed with variance $\pi^2/3n \approx 3.29/n$.

In this exercise, there is no closed-form solution to the likelihood equation; therefore, finding the maximum likelihood estimator requires some kind of numerical method. Newton's method is a logical choice: Letting $\ell(\theta)$ denote the loglikelihood function and starting from some value θ_0, let

$$\theta_{i+1} = \theta_i - \frac{\ell'(\theta_i)}{\ell''(\theta_i)} \tag{6}$$

for $i = 0, 1, \ldots$ until the value of θ_i stops changing. An interesting theoretical result is that if we take θ_0 to be a $\sqrt{n}$-consistent estimator like the sample median, then θ_1 itself is an asymptotically efficient estimator, just like the MLE $\hat{\theta}$. This is how δ is defined in equation (5), which is why δ may be called a one-step estimator.

It would be instructive to supplement this exercise in order to introduce students to the technique of Fisher scoring, in which $\theta_{i+1} = \theta_i + 3\ell'(\theta_i)/n$ for this example. This formula arises when $\ell''(\theta_i)$ of equation (6) is replaced by $-nI(\theta)$, where $I(\theta)$, the Fisher information, equals $1/3$ in this case. Like the one-step Newton-method estimator of equation (5), the one-step scoring estimator is asymptotically efficient.

The main computing benefit of this exercise is that it forces students to implement Newton's method. Because the same basic algorithm used to compute δ may be iterated to find the MLE, students should be encouraged to write a program that could be used for either purpose: Newton's method could be iterated only once or repeatedly until convergence, depending on a user-settable option. This subtle point encourages good programming practice as opposed to a "brute-force" approach in which students write two separate, inflexible programs with substantial overlap.

Table 1 exhibits three of the 1000 samples I generated to complete this exercise, along with the corresponding values of $\tilde{\theta}$, $\hat{\theta}$, and δ. We see that the sample variances are a bit lower than the $4/15 = 0.267$ for the median and $3/15 = 0.200$ for the MLE and one-step estimator predicted by the asymptotic approximation. Nonetheless, the values are pretty close; as a related activity,

Sample number	Simulated Data ($n = 15$)	Median $\tilde{\theta}$	MLE $\hat{\theta}$	One-step δ
1	-1.456 0.257 0.376 0.356 2.699 4.461 1.728 3.479 4.312 3.883 2.863 -0.640 2.364 -0.317 3.092	2.364	1.961	1.964
2	0.451 -3.310 0.104 2.431 0.077 1.567 -0.111 1.981 2.360 2.223 2.175 4.002 4.807 3.096 2.800	2.175	1.831	1.831
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1000	1.452 5.630 0.623 1.351 -3.126 4.681 0.924 0.702 0.976 3.137 2.083 2.158 -0.814 2.691 1.205	1.351	1.545	1.545
	Sample variances:	0.2347	0.1870	0.1871

Table 1: Three of the 1000 samples are shown, along with three different estimators of θ for each sample.

students could be asked to report approximate confidence intervals for the true variances to see whether the asymptotically predicted values fall within the intervals.

Glancing at the last two columns of Table 1 reveals an insight that is not implied by the asymptotic theory: When Newton's method is started at a reasonable estimate of θ, such as the median in this case, a single iteration of Newton's method winds up very close to the MLE. In fact, the sample mean of $|\delta - \hat{\theta}|$ for these data is only 0.00077. Coupled with the asymptotic theory showing that δ is essentially just as good an estimator as $\hat{\theta}$, this bit of empirical evidence helps the student understand that the one-step estimator is an attractive alternative to the MLE from a computational standpoint. Because the asymptotic theory in this example blends so nicely with the computing, each informing the other, a golden opportunity is missed if students are asked to learn the theory of these one-step estimators without performing any computation.

5 Discussion

The premise of this article is that computing topics can and should be taught throughout the graduate statistics curriculum, when possible. It is not my claim that such teaching should supplant courses devoted exclusively to statistical computing topics, nor do I delude myself by assuming that all instructors of theoretical courses will find it easy to integrate computing topics into these courses. Nonetheless, when the material in a theoretical course and the expertise of the instructor permit it, I believe a golden opportunity is lost when computing is not integrated into the course. Computing skills — including but not limited to facility with statistical software — are becoming more and more important to statisticians, and we fail to adapt the training of our graduate students to this reality at our own peril.

The three example exercises presented in this article demonstrate the useful symbiosis that occurs when computing is thoughtfully incorporated into a theoretical statistics course. There are several means by which such exercises may be created: As instructors, we should watch for things like theoretical results that lend themselves to simulation; large-sample results that may be checked for fixed small sample sizes; problems with closed-form solutions that may be checked numerically, or problems without closed form solutions that may be solved numerically; and statistical methods that may be illustrated using data. All of the exercises here can be completed using almost any statistical software and a bit of ingenuity. Even low-level languages like C and FORTRAN may be used, though in those cases it may be necessary to rely on already-written software to handle tasks such as uniform random number generation and graphical plotting. Finally, symbolic computation environments like Mathematica or Maple can be very useful for simplifying complicated expressions or obtaining numerical approximations; I find that one or two classroom demonstrations of such environments are generally enough to pique students' interest enough so that they learn and begin to use them in a rudimentary way on their own.

One danger in adopting the approach espoused in this article is that the all-important theoretical content of a course can be overshadowed if computing becomes too central. The instructor alone can ensure that this does not happen. For instance, if students are repeatedly required to write complicated programs from scratch, the focus of the course can shift to programming at the expense of theory. One method that can be used to mitigate against this particular drawback is for the instructor to provide skeleton computer code that students can modify. Particularly if such code is well-commented, valuable computing lessons can be learned without distracting from the main theme of the course.

Ultimately, computing exercises should not merely be created haphazardly; some thought should go into their pedagogical value. Such a focus on pedagogy, applied to the implementation of computing assignments, might even spread to all aspects of the course — not a bad outcome!

References

Ferguson, T. S. (1982), An inconsistent maximum likelihood estimator, *Journal of the American Statistical Association*, **77**: 831–834.

Gentle, J. E. (2004), Courses in statistical computing and computational statistics, *The American Statistician*, **58**: 2–5.

Appendix

This section gives sample computer code written in the S language, as implemented in R and S-plus, for performing some of the tasks required in the three exercises explained in this article. Though this code was written for R, most of it also works in S-plus (an exception is the "rmultinom" function); in addition, S-plus has a function called "cdf.compare" not found in R that is useful for comparing the empirical cdf with a theoretical cdf as in the Pearson chi-square example.

Sample code for the inconsistent MLE exercise

```
# Define the functions delta(theta) and Hinv(x) as described
# in Section 2:
delta <- function(theta) exp(-1/(1-theta)^4)
Hinv <- function(x) 2*cos(4*pi/3+acos(1-2*x)/3)

# To generate from a mixture distribution, define a vector of
# Bernoullis and a vector of uniforms:
z <- rbinom(n, 1, theta)
u <- runif(n)
# Now whenever z==0, transform to the translated and scaled
# h density and whenever z==1, transform to Uniform(-1,1).
u[z==0] <- theta + delta(theta) * Hinv(u[z==0])
u[z==1] <- u[z==1]*2-1

# To plot the loglikelihood, use the techniques described in
# Section 2 to carefully write a function loglik(theta, u) that
# will evaluate the loglikelihood CORRECTLY given a scalar theta
# and data vector u, no matter what the value of theta is.  Then:
thseq <- seq(0,1,len=400)[-c(1,400)]  # Omit 0 and 1 from sequence
thseq <- sort(c(thseq,u[u>0])) # add positive data to theta vector
y <- rep(0,length(thseq)) # Set up vector for loglikelihood values
for(i in 1:length(thseq)) {
   y[i] <- loglik(thseq[i], u)
}  # Now y may be plotted against thseq.
```

Sample code for the Pearson chi-square exercise

```
# First, generate the multinomials as a 3x1000 matrix:
mn <- rmultinom(1000, 50, c(.09, .42, .49))

# Next, create functions to compute W and X for a
# given multinomial vector n.
W <- function(n) {
   np <- sum(n)*c(.09, .42, .49)
   sum((n-np)^2/np)
}
# The definition of X will be similar to that of W

# Finally, compute a vector of chi-square statistics
# from the data.
chisqW <- apply(mn, 2, W)
# The definition of chisqX will be similar

# Graph the empirical cdf (see also "cdf.compare" in S-plus):
plot(ecdf(chisqW))

# To compare with the true chi-square on, say, 1 df:
```

```
x <- seq(0,max(chisqW),len=200)
lines(x, pchisq(x, 1))
```

Sample code for the one-step estimator exercise

```
# Define the derivative of the loglikelihood as a function
dl <- function (theta, x) {
  2*sum(1/(1+exp(theta-x))) - length(x)
}
# Next, define d2l, the second derivative, similarly

# Now a function that does a specified number of
# Newton iterations:
newton <- function(x, theta0, df, d2f, iterations=1) {
  theta <- theta0
  for(i in 1:iterations) {
    theta <- theta - df(theta, x) / d2f(theta, x)
  }
  theta
}

# If d2l is defined correctly, we can find the one-step
# estimator based on a sample from logistic with theta=2:
x <- rlogis(15)+2
newton(x, median(x), dl, d2l) # Use default iterations=1

# To get the MLE, just use a large number of iterations:
newton(x, median(x), dl, d2l, iterations=20)
```